1．从总体上说，“数据结构”课程主要研究数据的 逻辑结构、存储结构和算法 三个方面的内容。

    2．若对某线性表最常用的操作是在表中插入元素或者删除表中元素，则对于顺序存储结构和链式存储结构这两种存储结构而言，线性表应该采用 链式存储结构 。

    3．在长度为n的非空队列中进行插入或者删除操作的时间复杂度用大O符号表示为 O(1) 。

    4．若一棵度为4的树中度为1、2、3和4的结点个数分别为4、2、1和1，则该树中叶结点的个数为 8 。

    5．若某二叉树的中序遍历序列为B,A,F,D,G,C,E，按层次遍历序列为A,B,C,D,E,F,G，则该二叉树的后序遍历序列为 B,F,G,D,E,C,A 。

    6．将一棵结点总数为n、且具有m个叶结点的树转换为一棵二叉树以后，该二叉树中右子树为空的结点有 n-m+1 个。

    7．对于图G=(V,E) 与 G&acute;=(V&acute;,E&acute;)，若V&acute;包含于V，E&acute;包含于E，则称G&acute;是G的 一个子图 。

    8．在顺序表(6,15,30,37,65,68,70,72,89,99)中采用折半查找法查找元素37，与表中进行过比较的元素依次是 65,15,30,37 。     9．若已知n个关键字值具有相同的散列函数值，并且采用线性探测再散列法处理冲突，那么，将这n个关键字值全部散列到初始为空的地址空间中，发生散列冲突的次数是 n&#61620;(n-1)/2 。

10．若长度为n的序列K=(k1,k2,…,kn)当且仅当满足ki≤k2i并且ki≤k2i+1(1≤i≤n/2)时，则称该序列为一个小顶堆积(Heap)。根据该定义，序列(26,5,77,1,61,11,59,48,15,19)对应的小顶堆积是 1,5,11,15,19,77,59,48,26,61 。

二、简答题(本题共20分，每小题各5分)

    1．答：该邻接矩阵是稀疏矩阵。因为邻接矩阵中一共有10000个元素，只有200个非零元素，非零元素的数目只占整个稀疏矩阵元素总个数的2%，按照题目假设，可以认为该邻接矩阵是稀疏矩阵。

    2．答：该方法的基本思想是当发生散列冲突时按照下列方法求得后继散列地址： Di=H(k)+di) MOD m    i=1,2,…,n   (n≤m-1)

      其中，H(k)为散列函数，k为关键字，m为散列表长，di为地址增量序列，可以有以下三种取法：（1）di=1,2,3,…,m-1 称为线性探测再散列；

    （2）di=1^2,-1^2,2^2,-2^2,…,±n^2（n≤m/2）称为二次探测再散列；     （3）di=伪随机数序列，称为伪随机探测再散列。

    3．答：该结果是采用了起泡排序法排序得到的。若选择排序法每一趟排序选择一个最大值元素，该最大值元素只需要与当前参加排序的最后那个元素交换位置，而不需要改变其他元素的位置，显然，从上述结果可以看出不是如此。上述结果符合起泡排序的规律。

    4．答：最小递归深度为log2n+1，最大递归深度n。

三、综合题(本题共20分，每小题各5分)

    1．第④条语句有错，正确的语句是p->rlink->llink=p;

    2．解：若完全二叉树的第7层有10个叶结点，则有两种情况：

       ① 10个叶结点集中在第7层的最左边，此时可求出该二叉树的结点总数为(2^6-1)+10=73。

       ② 该完全二叉树的深度为8，10个叶结点集中在第7层的最右边，此时，可求出该二叉树的结点总数为(2^6-1)+2^7-1+108=235。

    因此，根据题意，该完全二叉树最多有235个结点。

    3．证明：设无向图的边数为e，顶点vi的度为TD(vi)，根据图的性质，有关系 2e=∑TD(vi)    （1≤i≤n）

    由于每一个顶点最多与图中其他n-1个顶点直接有关，即图中每一个顶点的度的最大值为n-1，因此，图中n个顶点的度的最大值之和为n(n-1)，即 2e=∑TD(vi)=n(n-1)   （1≤i≤n）     于是，有 e=n(n-1)/2     证毕

4．解：

    （注:每一趟选择最大者放前面）         （注:每一趟选择最小者放后面）

原 始：80,30,50,10,90,20 原 始：80,30,50,10,90,20

第1趟：90,30,50,10,80,20 第1趟：80,30,50,20,90,10

第2趟：90,80,50,10,30,20  第2趟：80,30,50,90,20,10

第3趟：90,80,50,10,30,20 第3趟：80,90,50,30,20,10

第4趟：90,80,50,30,10,20 第4趟：80,90,50,30,20,10

第5趟：90,80,50,30,20,10  第5趟：90,80,50,30,20,10

四、算法设计题（本题15分）

int TOPOTEST(TOPOVLink G[],vertype V[],int n)

 {   Elink \*p;

  int i,k;

       for(i=0;i<n;i++){

for(k=0;k<n;k++){

            if(G[k].vertex==V[i]){         /\* 若顶点V[i]是G中的顶点 \*/

 if(G[k].indegree!=0)         /\* 若顶点V[i]的入度不为0 \*/

                  return 0;                /\* 序不是该有向图的序列 \*/

 p=G[k].link;                /\* 若顶点V[i]的入度为0 \*/

while(p!=NULL){

                  G[p->adjvex].indegree--;  /\* 相关顶点的入度减1 \*/

 p=p->next;               /\* p指向下一个边结点 \*/

   }

               break;                      /\* 测试序列的下一个顶点 \*/

   }

}

}

     return 1;                             /\* 序列是该有向图的拓扑序列 \*/

}

五、单项选择题（本题共20分，每小题各2分）

    1．C   2．D   3．A   4．C   5．B   6．D   7．A   8．D   9．B   10．A

六、简答题（本题共20分，每小题各5分）

    1．答：① 通过头文件来调用库功能。在很多场合，源代码不便(或不准)向用户公布，只向用户提供头文件和二进制的库即可。用户只需要按照头文件中的接口声明来调用库功能，而不必关心接口怎么实现的。编译器会从库中提取相应的代码。

    ② 头文件能加强类型安全检查。如果某个接口被实现或被使用时，其方式与头文件中的声明不一致，编译器就会指出错误，这一简单的规则能大大减轻程序员调试、改错的负担。

    2．答：#include “filename.h”表明该文件是用户提供的头文件，查找该文件时从当前文件目录开始；#include <filename.h>表明这个文件是一个工程或者标准头文件，查找过程会检查预定义的目录。

    3．答：① 生命周期不同：

      &#8226; 全局变量随主程序创建和创建，随主程序销毁而销毁；       &#8226; 局部变量在局部函数内部，甚至局部循环体等内部存在，退出就不存在； 内存中分配在全局数据区。     ② 使用方式不同：

      &#8226; 通过声明后全局变量程序的各个部分都可以用到；

&#8226; 局部变量只能在局部使用。

    4．答：指针变量也占用内存单元，而且所有指针变量占用内存单元的数量都是相同的，就是说，不管是指向何种对象的指针变量，它们占用内存的字节数都是一样的，并且要足够把程序中所能用到的最大地址表示出来（通常是一个机器字长）。

七、填空题（本题共20分，每小题各2分）     1．① 50      ② x             2．① (-1)\*f ② 2\*i+1

    3．① c=c+5    ②c=c-21     4．① \*t      ② \*s-\*t

    5．① str3[k]=str2[j++]   ② str1[i]==„\0‟     6．① argc>1   ② \*argv

    7．① a+    ② r    ③ fp2     ④ fgetc(fp2)     8．统计正整数n的位数

    9．判断输入的字符串是否为“回文”

    10．以追加方式打开文件“file.txt”后向文件写入“data”，然后查看(输出)文件指针位置。

八、程序设计题（本题15分）    #include <stdio.h>

 #include <string.h>

   int func(char \*s,int n,char ch)

 {    int j,k=0;

 s[n]=ch;

s[n+1]=„\0‟;

 while(s[k]!=ch)

k++;

       if(k==n)

return 0; /\* 字符串中不存在该字符 \*/

  else{ /\* 字符串中存在该字符 \*/

for(j=k;j<n;j++)

s[j]=s[j+1];            /\* 删除首次出现的该字符 \*

s[j-1]=„\0‟;

  return k+1;       /\* 该字符在字符串中位置 \*/

}

}

main( )

{    char s[80],ch;

int len,position;

gets(s);

       puts(s);             /\* 输出删除前的字符串 \*/

 printf(“Input a char”);             /\* 输入字符串 \*/

 scanf(“%c”,&ch);             /\* 输入字符 \*/

       len=strlen(s);                              /\* 计算字符串的长度 \*/

position=fun(s,len,ch);

 if(position==0) printf(“Not exit!\n”);

printf(“Not exit!\n”);

else{

puts(s);                       /\* 输出删除后的字符串 \*/

printf(“\nPosition=%d\n”,position); /\* 输出位置 \*/

 }

 }